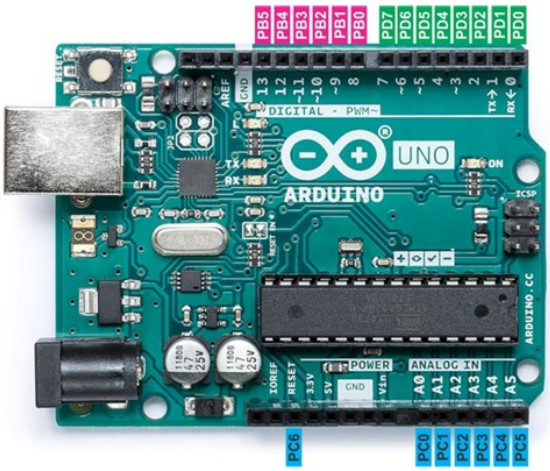
Allgemeine Imports

```
#define F_CPU 16000000  // setzt die Frequenz der CPU zu 16MHz
#include <util/delay.h>
#include <avr/io.h>
#include <avr/interrupt.h> // wird benötigt um Interrupts zu verwenden
#include <util/atomic.h> // wird benötigt um atomic-Blöcke zu nutzen
```

Arduino PIN Belegung

3 Register -> PORTB, PORTC, PORTD



Digital pins

0 – PD0
1 – PD1
2 – PD2
3 – PD3
4 – PD4
5 – PD5
6 – PD6
7 – PD7

Digital pins

8 – PB0
9 – PB1
10 – PB2
11 – PB3
12 – PB4
13 – PB5

Analog pins

A0 – PC0
A1 – PC1
A2 – PC2
A3 – PC3
A4 – PC4
A5 – PC5

DDR. – Datenrichtungsregister -> Output = 0 / Input = 1
PORT. – Datenregister -> nicht aktiv = 0 / aktiv = 1

wenn man spezielle Ports setzen will:

0	1	2	3	4	5	6
1						
0	1					
0	0	1				
0	0	0	1			
0	0	0	0	1		
0	0	0	0	0	1	
0	0	0	0	0	0	1
0x01	**0x02**	**0x04**	**0x08**	**0x10**	**0x20**	**0x30**

Bsp.

- `DDRD = 0xFF;` -> alle Ports auf Input
- `DDRD = 0x00;` -> alle Ports auf Output
- `PORTD |= (1<<PORTD1);` -> PD1 wird aktiv gesetzt (durch | wird Bit wirklich nur auf dieser Stelle verändert)
- `PORTD &= ~(1<<PORTD1);` -> PD1 wird negiert
- `DDRD |= 0x12;` -> PD2 und PD3 auf Input

Interrupts allgemein

volatile

- Compiler löscht Variable nicht, wenn er meint, dass diese nicht mehr verwendet wird
- zeigt, dass Variable jederzeit verändert werden kann
- wenn man Variablen in IR-Routinen verändert, diese mit volatile deklarieren
- `static volatile int buttonStatus = 0;`

Atomic functions

- verwenden um Interrupts in heiklen Situation zu verhindern
- damit Wert nicht während einer Berechnung von ausgelöstem Interrupt verändert werden kann
- #include <util/atomic.h>
- ATOMIC_FORECON => Interrupts nach Block immer aktiviert
- ATOMIC_RESTORESTATE -> Interrupts auf selben Status wie vor Block

```
int main(void)
{
   ctr = 0x200;
   start_timer();
   sei();
   uint16_t ctr_copy;
   do
   {
     ATOMIC_BLOCK(ATOMIC_FORCEON)
     {
       ctr_copy = ctr;
     }
   }
```

Enable interrupts

- mit sei() Methode
- aktiviert Interrupts durch Einstellen der globalen Interrupt-Maske

Disable Interrupts

- mit cli() Methode
- deaktiviert Interrupts durch Leeren der globalen Interrupt-Maske

Buttons

PORT an dem Button angeschlossen ist, muss auf Input gesetzt werden, da PORT standardmäßig auf 0 ist, nichts notwendig

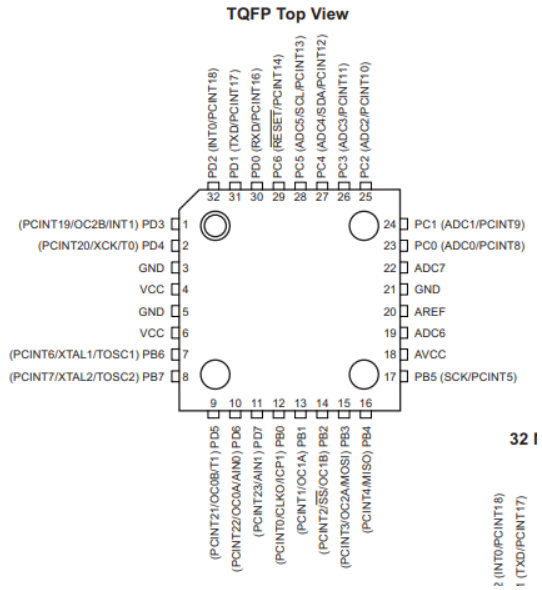
Internen Pull-Up Widerstand verwenden

1. PORT an dem Button angeschlossen ist auf HIGH setzen: -> `PORTC |= (1<<PORTC1);`
2. **PCINT** PORT finden -> **Hier**: PCINT9

1. Pin Configurations

Figure 1-1. Pinout

TQFP Top View



3. **Mask-Register** finden, um Interrupts an diesem Port zu aktivieren -> **Hier**: PCMSK1

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

4. **Pin-Change Interrupt** auswählen -> PCIE1 = irgendeine Veränderung bei PCINT 8 bis 14 -> **Hier**: PCICR = PCIE1
   - **PCIE0:** Pins 0-7
   - **PCIE1:** Pins 8-14
   - **PCIE2:** Pins 16-23

12.2.4 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

5. Interrupt Vektor auswählen -> **Hier**: PCINT1
   - **PCINT0_vect:** Pins 0-7
   - **PCINT1_vect:** Pins 8-14
   - **PCINT2_vect:** Pins 16-23

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A

6. Interrupt konfigurieren -> `sei();` und Vektor-Routine: `ISR(PCINT1_vect){}`
7. IR-Routine -> überprüfen ob Button gedrückt -> `if (!(PINC & (1 << PINC1))){}`

Fertige Konfiguration:

```
DDRC |= ~(1<<PORTC1); // Port auf Input

PORTC |= (1<<PORTC1); // Port auf HIGH setzen

//Button-Interrupts
PCMSK1|=(1<<PCINT9);   //erlaubt PCINT9 einen Interrupt auszulösen
PCICR |=(1<<PCIE1); //festgelegt, wann Interrupt aufgerufen wird PCIE1 = any change on any PCINT14..8

sei();

ISR(PCINT1_vect) //PIN Change Interrupt Routine, 1 legt Priorität fest
{
    if (!(PINC & (1 << PINC1)))
    {

    }
}
```

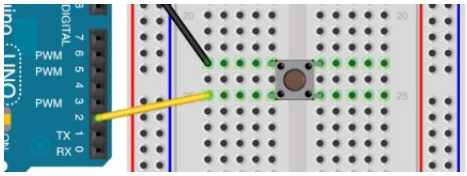
Pull-Down Widerstand verwenden

Wie vorher, nur muss PORT auf „nicht aktiv“ gesetzt werden und IR-Routine sieht so aus:

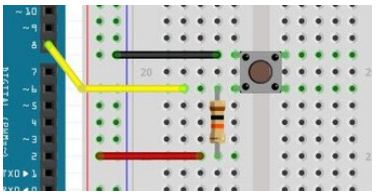
```
ISR(PCINT1_vect) //PIN Change Interrupt Routine, 1 legt Priorität fest
{
    if ((PINC & (1 << PINC1)))
    {

    }
}
```

Um prellen von Buttons zu vermeiden -> Delay einbauen.

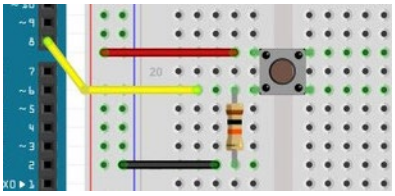
Internen Pull Up Widerstand verwenden:



Externen Pull Up verwenden:



Externen Pull Down verwenden:



LCD

```
#include "lcd.h"
```

wichtige Funktionen von LCD Library:

- Display einschalten -> `lcd_init(LCD_DISP_ON);`
- Display leeren -> `lcd_clrscr();`
- String ausgeben -> `lcd_puts(“a”);`
- Char ausgeben -> `lcd_putc(‘a’);`
- Cursor auf 0,0 setzen -> lcd_home();
- Cursor setzen -> lcd_gotoxy(0,1);

Verwendete PORTS

PORTC 1-3, PORTD 4-7 für Datenleitung

ADC

ADCSRA - ADC Control and Status Register

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADEN -> ADC einschalten

ADIE -> ADC löst Interrupt nach Wandlung aus

ADSP. -> ADC Prescaler wählen

ADPS2	ADPS1	ADPS0	Vorteiler
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADMUX

ADSC -> Start einer Wandlung

REF0 - Spannungsreferenz -> interne Referenz Avcc

MUX. -> welcher analoge Eingang mit ADC verbunden ist

MUX3	MUX2	MUX1	MUX0	Kanal
0	0	0	0	Kanal 0, Pin PC0
0	0	0	1	Kanal 1, Pin PC1
0	0	1	0	Kanal 2, Pin PC2
0	0	1	1	Kanal 3, Pin PC3
0	1	0	0	Kanal 4, Pin PC4
0	1	0	1	Kanal 5, Pin PC5
0	1	1	0	Kanal 6 (*)
0	1	1	1	Kanal 7 (*)
1	1	1	0	1.23V, Vbg
1	1	1	1	0V, GND

Bsp.

```
ADCSRA |= (1<<ADEN) | (1<<ADIE) | (1<<ADPS0) | (1<<ADPS1) |(1<<ADPS2);

ADMUX |= (1<<REFS0) | (1<<MUX0) |(1<<MUX2);

ADCSRA |= (1<<ADSC); //Wandlung starten

ISR(ADC_vect){
	//Wandlung auslesen
	resultVolt = ADCW;
```

```
        ADCSRA |= (1<<ADSC); //erneute Wandlung starten
}
```

Timer

Es gibt 8-bit Timer -> TCCR0A und 16-bit Timer -> TCCR1A

CTC Timer

- TCCR0A auf WGM01
- `TCCR0A |=(1<<WGM01);`

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Prescaler

- TCCR0B
- `TCCR0B |=(1<<CS01) | (1<<CS00);`

Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Output Compare Register

- Wird festgelegt wann Timer Counter value Interrupt auslöst
- `OCR0A = 250;`

Timer Mask setzen

- Um festzulegen, welcher OCR0A verwendet werden soll
- `TIMSK0 |=(1<<OCIE0A);`

- **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**

When the OCIE0B bit is written to one, and the I-bit in the status register is set, the Timer/Counter compare match B interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter interrupt flag register – TIFR0.

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the status register is set, the Timer/Counter0 compare match A interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 interrupt flag register – TIFR0.

Interrupt-Routine

```
ISR(TIMER0_COMPA_vect)
{
}
```